International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 10 | Oct -2017 www.irjet.net p-ISSN: 2395-0072

The Study of the Large Scale Twitter on Machine Learning

ABHISH IJARI

Dept of Information Science and Engineering, KLEIT, Hubballi, Karnataka, India

---***---

Abstract – *The major rate of data-driven in success of finding solutions to different problems, The amount of data with dropping costs and processing that has led in large scale machine. The work on this project on a study of Twitter's integration tools with that on a tool into an existing Hadoop based Pig-centric platform. The main role on this platform handles business intelligence task and data warehousing. The core of this work exist in mainly recent extensions of Pig in provide predictive analytics capabilities that incorporate machine learning, specifically focused on super-vised classification. Thus on a scale it has identified technique of a stochastic gradient for learning online and highly amenable for scaling out large amount of data. The machine learning enables success in an sample of a data on an training and testing which achieves directly in Pig, because an crafted loaders and storage function can Pig script it integration. With an existing infrastructure, screening out of a production environment will become libraries for script in material output.*

Key Words: **stochastic gradient descent, online learning, ensembles, logistic regression**

1. INTRODUCTION

Hadoop, the open-source implementation of Map Reduce [15], has emerged as a popular framework for large-scale data processing. Among its advantages are the ability to horizontally scale to petabytes of data on thousands of commodity servers, easy-to-understand programming. With high degree of fault tolerance. Although originally designed for applications such as text analysis, web indexing, and graph processing, Hadoop can be applied to manage structured data as well as \dirty" semi structured datasets with inconsistent schema, missing ends, and invalid values. More schematically the termed predictive analysis on fundamental contribution in a case study can experience a generalized organisation that exploit a new breed of engineers known as data scientists.

Today, Hadoop enjoys widespread adoption in organizations ranging from two-person start-ups to Fortune 500 companies. It lies at the core of a software stack for large-scale analytics, and owes a large part of its success to a vibrant ecosystem. For example, Pig [37] and Hive [47] provide higher-level languages for data analysis: a data ow language called Pig Latin and a dialect of SQL, respectively. H-Base the open source implements of Google's Big-Table [13], provides a convenient data model.

1.1 Descriptive Statistics

The value of a Hadoop-based stack for \traditional" data warehousing and business intelligence tasks has already been demonstrated by organizations such as Facebook, LinkedIn, and Twitter (e.g., [22, 41]). This value proposition also lies at the centre of a growing list of start-ups and large companies that have entered the \big data" game. Common tasks include ETL, joining multiple disparate data sources, followed by altering, aggregation, or cube materialization. Statisticians might use the phrase descriptive statistics to describe this type of analysis. These outputs might feed report generators, frontend dashboards, and other visualization tools to support common \roll up" and \drill down" operations on multi-dimensional data. Hadoop-based plat-forms have also been successful in supporting ad hoc queries by a new breed of engineers known as \data scientists". The success of the Hadoop platform drives infrastructure developers to build increasingly powerful tools, which data scientists and other engineers can exploit to extract insights from massive amounts of data. In particular, we focus on machine learning techniques that enable what might be best termed predictive analytics. We readily acknowledge that this paper does not present any fundamental contributions to machine learning. Rather, we focus on end-to-end machine learning workflows and integration issues in a production environment. Although specifically a case study, we believe that these experiences can be generalized to other organizations and contexts, and therefore are valuable to the community.

2. BACKGROUND AND RELATED WORK

We begin with a brief overview of machine learning. Let X be the input space and Y be the output space. Given set of training samples D = f(x_1; y_1); (x_2; y_2):::(x_n; y_n)g from the space X Y (called labelled examples or instances), the supervised machine learning task is to induce a function f : X ! Y that best explains the training data. The notion of \best" is usually captured in terms of minimizing \loss", via a function L which quantities the discrepancy between the functional prediction f(x_i) and the actual output y_i, for details, we refer the reader to standard textbooks. There are three main components of a machine learning solution: the data, features extracted from the data, and the model. Accumulated experience over the last decade has shown that in real-world settings, the size of the dataset is the most important factor. Studies have repeatedly shown that simple models trained over enormous quantities of data outperform more sophisticated models trained on less data. Labelled training examples derive from many sources. Human annotators can be paid to manually label examples, and with the advent of crowdsourcing the cost can be quite reasonable [45]. However, the amount of training data that can be manually generated pales in comparison to the amount of data that can be extracted automatically from logs and other sources in an organization's data warehouse. As a simple example, query and interaction logs from commercial search engines can be distilled into relevance judgments [24]. These data tend to be noisy, but modern \learning to rank" [27] algorithms are resilient (by design) to noisy data. By mining log data, an organization can generate practically limitless amounts of training data for certain tasks.

Despite growing interest in large-scale learning, there are relatively few published studies on machine learning works and how such tools integrate with data management platforms: Scullery et al. [42] describe Google's e orts for detecting adversarial advertisements. Cohen et al. [14] advocate the integration of predictive analytics into traditional RDBMSes. Labelled training examples derive from many sources. Human annotators can be paid to manually label examples, and with the advent of crowdsourcing the cost can be quite reasonable [45]. However, the amount of training data that can be manually generated pales in comparison to the amount of data that can be extracted automatically from logs and other sources in an organization's data warehouse.
Despite growing interest in large-scale learning, there are relatively few published studies on machine learning work- flows and how such tools integrate with data management.

3. TWITTER'S ANALYTICS STACK

A large Hadoop cluster lies at the core of our analytics infrastructure, which serves the entire company. Data is written to the Hadoop Distributed File System (HDFS) via a number of real-time and batch processes, in a variety of formats. These data can be bulk exports from databases, application logs, and many other sources. When the contents of a record are well they are serialized using either Protocol Bu ers[3] or Thrift.[4] Ingested data are LZO-compressed, which provides a good trade between compression ratio and speed.

In a Hadoop job, different record types produce different types of input key-value pairs for the mappers, each of which requires custom code for deserializing and parsing. Since this code is both regular and repetitive, it is straightforward to use the serialization framework to specify the data schema, from which the serialization compiler generates code to read, write, and manipulate the data. This is handled by our sys-tem called Elephant Bird,[5] which automatically generates Hadoop record readers and writers for arbitrary Protocol Buffer and Thrift messages.

4. EXTENDING PIG

The previous section describes a mature, production system that has been running successfully for several years and is critical to many aspects of business operations. In this section, we detail Pig extensions that augment this data analytics platform with machine learning capabilities.

4.1 DEVELOPMENT HISTORY

To better appreciate the solution that we have developed, it is perhaps helpful to describe the development history. Twitter has been using machine learning since its earliest days. Surmise, a two year old startup that Twitter acquired primarily for its search product in 2008, had as part of its technology portfolio sentiment analysis capabilities based in part on machine learning. After the acquisition, machine learning contributed to spam detection and other applications within Twitter. These activities predated the existence of Hadoop and what one might recognize as a modern data analytics platform. Typically, data manipulation in Pig is followed by invocation of the machine learning tool (via the command line or an API call), followed perhaps by more data manipulation in Pig. During development, these context switches were tolerable (but undoubtedly added friction to the development process). In production, however, these same issues translated into brittle pipelines. It is common to periodically update models and apply classifiers to new data, while respecting data dependencies.

There are many issues with this work ow, the foremost of which is that down sampling largely defeats the point of working with large data in the rest place. Beyond the issue of scalability, using existing machine learning tools created. Typically, data manipulation in Pig is followed by invocation of the machine learning tool (via the command line or an API call), followed perhaps by more data manipulation in Pig. During development, these con-text switches were tolerable (but undoubtedly added friction to the development process). In production, however, these same issues translated into brittle pipelines.

4.2 CORE LIBRARIES

Our machine learning framework consists of two components: a core Java library and a layer of lightweight wrappers that expose functionalities in Pig. The core library is worth a passing description, but is not terribly interesting or innovative. It contains basic abstractions similar to what one might and in Weka, Mallet, Mahout, and other existing packages. We have a representation for a feature vector, essentially a mapping from strings to floating point feature values, mediated by integer feature ids for representational compactness. A classifier is an object that implements a classify method, which takes as input a feature vector and outputs a classification object (encapsulating a distribution over target labels). There are two different interfaces for training classifiers: Batch trainers implement a builder pattern and expose a train method that takes a collection of (label, feature vector) pairs and returns a trained classifier. Online learners are simply classifiers that expose an update method, which processes individual (label, feature vector) pairs. Finally, all classifiers have the ability to serialize their models to, and to load trained models from abstract data streams (which can be connected to local les, HDFS les, etc.).

Our core Java library contains a mix of internally built classifiers and trainers (for logistic regression, decision trees, etc.), as well as adaptor code that allows us to take advantage of third-party packages via a unified interface. All of these abstractions and functionalities are fairly standard and should not come as a surprise to the reader.

Our machine learning algorithms can be divided into two classes: batch learners and online learners. Batch learners require all data to be held in memory, and therefore the Pig storage functions wrapping such learners must first internally buffer all training instances before training. This presents a scalability bottleneck, as Hadoop reduce tasks are typically allocated only a modest amount of memory. Online learners, on the other hand, have no such restriction: the Pig storage function simply streams through incoming instances.

4.3 TRAINING MODELS

For model training, our core Java library is integrated into Pig as follows: feature vectors in Java are exposed as maps in Pig, which we treat as a set of feature id (int) to feature value (float) mappings. Thus, a training instance in Pig has the following schema:

One of the primary challenges we had to overcome to enable Pig-based machine learning was the mismatch between typical Pig data flows and data flows in training machine learning models. In typical Pig scripts, data ow from sources (HDFS les, HBase rows, etc.), through transformations (joins, alters, aggregations, etc.), and are written to sinks (other HDFS les, another HBase table, etc.). In this data ow, UDFs might read \side data", for example, loading up dictionaries. When training a classifier, the input data consist of (label, feature vector) pairs and the output is the trained model. The model is not a \transformation" of the original data in the conventional sense, and is actually closer (both in terms of size and usage patterns) to \side data" needed by UDFs.

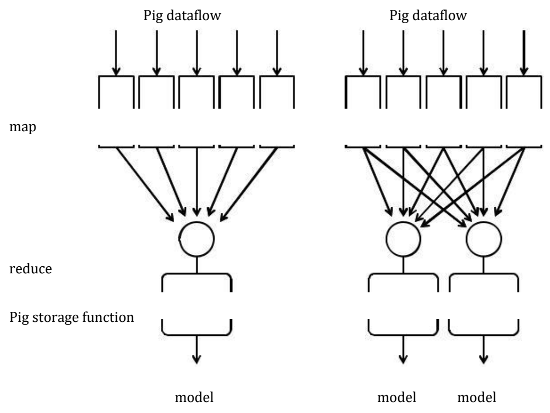

Figure 1: Illustration of how learners are integrated into Pig storage functions. By controlling the number of reducers in the final Map Reduce job, we can control the number of models constructed: on the left, a single classifier, and on the right, a two-classifier ensemble.

In short, the parallelization provided by running multiple reducers corresponds naturally to training ensembles of classifiers, and using this mechanism, we can arbitrarily scale out (overcoming the bottleneck of having to training data onto a single machine). As is often the case for many machine learning problems, and confirmed in our experiments (see Section 6), an ensemble of classifiers trained on partitions of a large dataset outperforms a single classifier trained on the entire dataset (more precisely, lowers the variance component in error). We have thus far not explicitly addressed feature generation, because it is largely dependent on the problem domain and requires the creativity of the engineer to be able to cull the relevant signals from vast quantities of data. However, we describe a sample application in Section 6 to give the reader a feel for the complete process.

5. SCALABLE MACHINE LEARNING

The head of machine learning is incredibly rich and diverse, but from the vast literature, we have identified two classes of techniques that are particularly amenable to large-scale machine learning. The rest is stochastic gradient descent, representative of online learners that can easily scale to large datasets. The second is ensemble methods, which allow us to parallelize training in an nearly embarrassingly parallel manner, yet retain high levels of effectiveness. Both are well known in the machine learning literature, and together they form a powerful combination. These techniques occupy the focus of our implementation efforts.

6. SENTIMENT ANALYSIS APPLICATION

In this section, we present an application of our machine learning tools to the problem of sentiment analysis. Although the problem is intrinsically interesting, our discussion primarily exists to illustrate the various features of our machine learning framework and show how all the pieces \come together".

6.1 METHODOLOGY

Sentiment analysis, and more broadly, opinion mining, is an area of natural language processing that has received significant interest in recent years; Pang and Lee provide a nice overview [40]. These technologies have also seen widespread commercial interest, particularly as applied to social media: sentiment analysis and related technologies promise solutions to brand and customer relations management, as well as insights into consumer behaviour in the marketplace. Sentiment analysis applied to tweets has naturally received attention [38, 36, 25]. In contrast to previous approaches, which use some form of linguistic processing, we adopt a knowledge-poor, data-driven approach. It provides a base-line for classification accuracy from content, given only large amounts of data.

More specifically, we tackle the binary polarity classification task. That is, given a tweet known in advance to express some sentiment, the classifier's task is to predict y_i 2f Negative; Positive g. To generate labelled training data for polarity classification, we use the well-known \emoticon trick". That is, we simply assume that tweets with positive emoticons, e.g., :-) and variants, are positive training instances, and tweets with negative emoticons, e.g., :-(and variants, are negative training instances. Obviously, these assumptions are not completely valid and do not capture phenomena such as sarcasm, irony, humour, etc., but, overall, data gathered in this manner are quite reasonable.

The illustrative purposes, the \emoticon trick" is typical of a mechanism for generating a large number of labelled, albeit noisy, training examples. We have a representation for a feature vector, essentially a mapping from strings to floating point feature values, mediated by integer feature ids for representational compactness. A classifier is an object that implements a classify method, which takes as input a feature vector and outputs a classification object (encapsulating a distribution over target labels). There are two different interfaces for training classifiers: Batch trainers implement a builder pattern and expose a train method that takes a collection of (label, feature vector) pairs and returns a trained classifier. Online learners are simply classifiers that expose an update method, which processes individual (label, feature vector) pairs.

7. CONCLUSIONS

As the cost of storage and processing continues to drop, organizations will accumulate increasing amounts of data from which to derive insights. Inevitably, the sophistication of analyses will increase over time. Business intelligence tasks such as cubing to support \roll up" and \drill down" of multi-dimension data are already commonplace, with mature best practices both in the context of traditional data warehouses and Hadoop-based stacks. We are, however, witnessing the transition from simple descriptive analytics to more powerful predictive analytics, which promises to unlock greater troves of insights. There has not yet emerged a consensus on architectures and best practices for these types of activities. Nevertheless, we hope that through the accumulation of experiences, the community will converge on a body of shared knowledge. We hope that our experiences in integrating ma-chine learning tools in a Pig-centric analytics environment contribute to this end goal. In addition to training models, many other types of data. The manipulations common in machine learning can be straight- forwardly accomplished in Pig. For example, it is often desirable to randomly shuffle the labelled instances prior to training, especially in the case of online learners, where the learned model is dependent on the ordering of the examples. This can be accomplished by generating random numbers for each training instance. Using our machine learning framework is mostly a matter of learning a few Pig idioms; the rest feels just like typical analysis tasks. Put it another way: machine learning becomes a natural extension of data science, where insights gleaned from data are operationalized in computational models.

ACKNOWLEDGEMENT

I would like to express thankful to Mr. Anand Mannikeri, Assistant Professor KLEIT Hubballi for technical suggestions and continuous support to complete this work.

REFERENCES

[1] K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and E.Paulson. E cient processing of data warehousing quiries in a split execution environment. SIGMOD, 2011.

[2] M. Banko and E. Brill. Scaling to very very large corpora for natural language disambiguation. ACL, 2001.

[3] R. Bekkerman and M. Gavish. High-precision phrase-based document classi cation on a modern scale. KDD, 2011.